Львівський національний університет імені Івана Франка
Факультет прикладної математики та інформатики
Кафедра дискретного аналізу та інтелектуальних систем

ДИПЛОМНА РОБОТА

На тему:

Створення додатка з використанням Windows App SDK для автоматизації та збереження повторюваних завдань операційної системи Windows

Студента IV курсу, групи ПМі-45
напряму підготовки комп’ютерні науки
Манецький О. В.

Науковий керівник: ас. Стойко Т. І.

Львів - 2023

ВСТУП

Кожен розробник програмного забезпечення знає і майже щодня використовує командний рядок для виконання різних команд. Зазвичай це скрипти для збирання та запуску проекту, запуску тестування, відвантаження та розгортання нової версії продукту та ще десятки інших. Так як необхідність у виконанні таких дій є майже щодня, усі ці команди потрібно пам’ятати або кожного разу шукати у документаціях проекту. Крім того, часто такі операції складаються з десятків команд. Наприклад, процес підготовки середовища для розробки це зазвичай велика інструкція, яка включає в себе багато кроків, адже потрібно виконати збирання проекту, встановити різні змінні середовища, завантажити необхідні інструменти для розробки, репозиторій, бібліотеки та пакети тощо. Не виконання однієї з команд швидше за все призведе до виникнення помилок надалі.

Завданням дипломної роботи є дослідження проблеми та створення додатку для збереження таких повторюваних команд. Головна ціль – скоротити час на виконання таких завдань та надати зручний користувацький інтерфейс для управління збереженими командами, їх запуск, створення нових, і що також важливо, зручний спосіб поширення готових команд для інших користувачів.

Зазначимо, що подібних та уже доступних додатків для Windows не існує, що і наштовхнуло на дослідження цієї теми. Схожий за ідеєю застосунок створила компанія Apple для своїх операційних систем MacOS та iOS (див. додаток А). Проте, його функціонал є більш орієнтований на звичайних користувачів аніж розробників, адже дозволяє створювати команди, що складаються з простих дій таких як «Надіслати повідомлення», «Відкрити програму». Тим не менш застосунок від Apple може бути концептуальним прототипом нашого додатка, але для іншої цільової аудиторії – розробників програмного забезпечення.

У рамках дослідження цієї теми також будуть розглянуті можливості та потенціал Windows App SDK, як нової платформи від Microsoft для створення сучасних десктопних застосунків.

1 ПОСТАНОВКА ЗАДАЧІ

Завданням дипломної роботи є створення застосунку для комп’ютерів, які працюють на операційній системі Windows. Розроблене програмне забезпечення повинне відповідати наступним вимогам:

1. Можливість створення та запуску команд, які місять різні дії;
2. Підтримка таких дій як «Виконати Command Prompt команду», «Виконати PowerShell команду» та інші;
3. Зручний та швидкий користувацький інтерфейс;
4. Можливість групування команд у різні групи (наприклад для різних проектів);
5. Функціонал поширення команд для інших користувачів.

Також вимогою є використання Windows App SDK та компонентів WinUI, адже це найбільш передовий та рекомендований підхід до створення десктопних застосунків, який пропонує компанія Microsoft сьогодні.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

2.1 Windows App SDK

Windows App SDK – найновіша на сьогодні платформа для розробки Windows додатків. Вона включає в себе уніфіковане API, компоненти та інструменти для розробки додатків під Windows 11 та Windows 10 (версій від 1809) [2].

При дослідженні теми Windows App SDK дуже часто трапляються згадки WinUI, UWP, WinRT, Project Reunion, MSIX і інших технологій, які спочатку заплутують розробника, який не має великого досвіду у розробці Windows додатків. Для повноти дослідження теми і розуміння специфіки розробки потрібно розібратись у вище згаданих технологіях.

2.1.1 Win32 та UWP додатки

Зробимо кілька кроків назад і подивимось на історію розвитку UI-бібліотек від компанії Microsoft (див рис. 2.1).

Раніше Windows додатки створювали з допомогою WinForms та WPF. Але з появою Windows 10 компанія Microsoft показала Universal Windows Platform (UWP), яка стала новим стандартом для розробки і поділила бібліотеки на дві групи: Win32 та UWP додатки. Основна різниця полягає у API, яке використовується для розробки. UWP використовує WinRT, що є сучасною Windows API, його інколи ще називають UWP API, а Win32 (або Desktop) додатки – Win32 API [7]. Нове API для UWP дозволяє запускати розроблені додатки не лише на персональних комп’ютерах, а і інших пристроях, що працюють на основі Windows: Xbox, HoloLens, Surface Hub та інші [4].

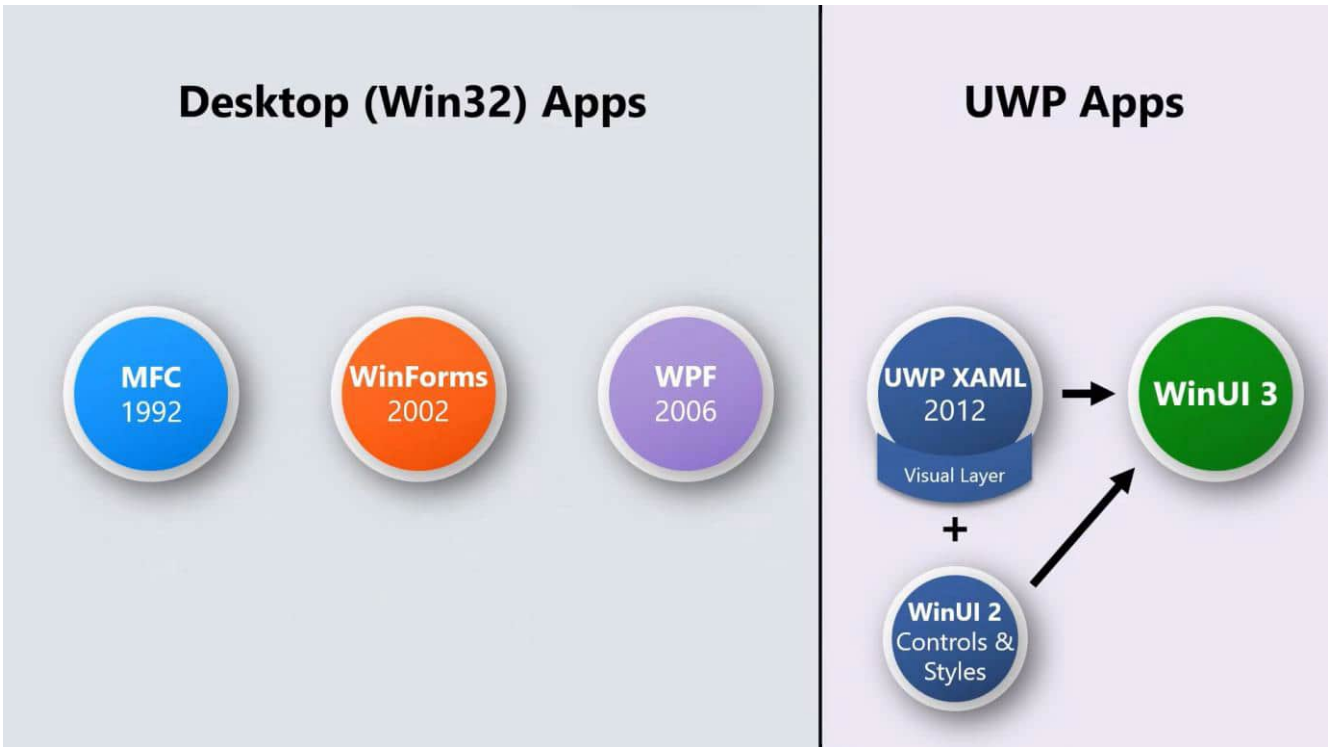

Рисунок 2.1– Еволюція UI-фреймворків від компанії Microsoft.

Проте різниця не лише у API. UWP використовує нову модель розробки. Усі додатки запускаються у спеціальному контейнері (sandbox), де системні ресурси обмежені [7]. Це також було зроблено для того, аби програми могли поширюватись через Microsoft Store. Додаток повинен оголосити про використання певних ресурсів системи. Такий же підхід використовується у мобільних застосунках: користувач бачить повідомлення про те, що додаток намагається доступитись до певних ресурсів і може дозволити або обмежити це.

Ще одна відмінність, яка значною мірою також з’явилась через можливість поширення додатків у Microsoft Store, це новий спосіб упакування застосунків, який має розширення *.appx* і, в загальному, базується на *.zip* форматі [7]. Система Windows знає, як встановлювати та видаляти додаток запакований таким чином. Головною перевагою такого підходу є те, що після видалення застосунку у системі не залишається безладу.

Також, важливо зазначити, що UWP використовує XAML технологію для створення користувацького інтерфейсу, яка базується на XML-розмітці [4]. XAML вперше з’явився при розробці на WPF і зберігся у UWP під назвою UWP XAML.

2.1.2 WinUI

Тепер коли ми дізнались що таке UWP можна перейти до WinUI і розібратись для чого вона створена.

Компоненти та стилі UWP XAML є частиною Windows. Це створює сильну залежність від операційної системи. Користувач не може користуватись останньою версією додатку, якщо його операційна система також не оновлена до останньої версії. Для того, аби позбутись такої залежності Microsoft виокремила компоненти у NuGet-пакет під назвою WinUI 2.0. [7]

Фреймворк UWP XAML також є частиною операційної системи і, відповідно, став такою ж небажаною залежністю. Тому його також було відокремлено у окремий пакет і об’єднано з WinUI 2.0. Таким чином з’явився новий NuGet-пакет з назвою WinUI 3.0. [7]

Пакет WinUI 3.0 є вже повноцінним самостійним фреймворком, який працює на основі UWP XAML фреймворку та компонентах. Важливим нововведенням є те, що він може використовуватись з моделями розробки UWP і Win32 (див рисунок 2.2 та 2.3). [3]

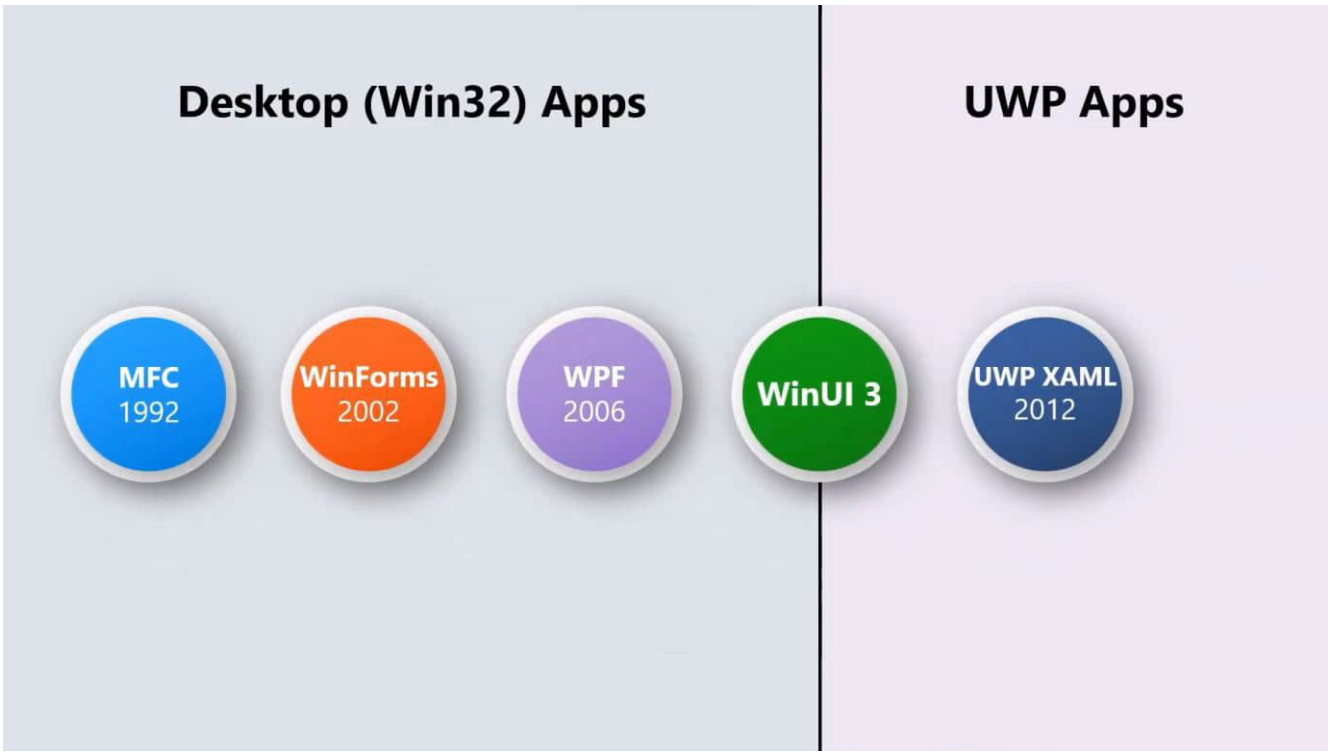

Рисунок 2.2 – UI-фреймворки компанії Microsoft.

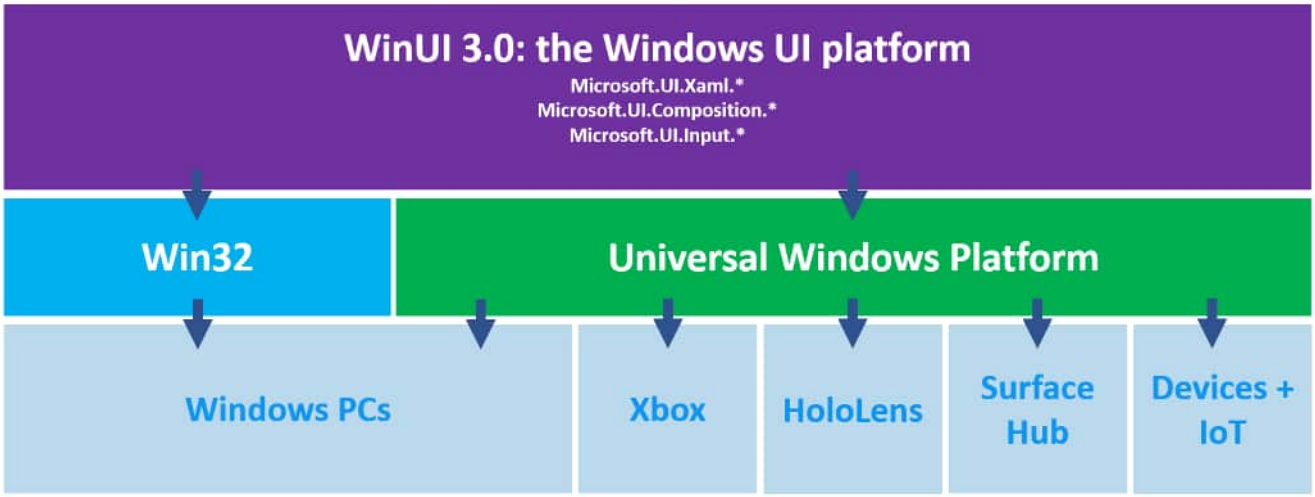

Рисунок 2.3 – Моделі розробки використовуючи бібліотеку WinUI 3.0

2.1.3 MSIX

У пункті 2.1.1 уже згадувалось про новий спосіб упакування для UWP додатків. Операційна система знає, як встановлювати такі додатки і видаляти їх не залишаючи залишкових файлів. Так як технологія упакування вбудована у нові версії Windows, цю функціональність також можна використовувати для Win32

додатків. Для застосунку створюється MSIX-пакет, який також є стандартним форматом упакування Windows. Файл, який отримується в результаті пакування має розширення *.msix* і є тим самим форматом що і *.appx* для UWP. MSIX-пакети (як і APPX) можна поширювати через Microsoft Store.

2.1.4 Project Reunion

Раніше уже згадувалось, що UWP додатки використовують Windows Runtime (WinRT), що є сучасним Windows API, яке доступне з версії Windows 10 та підтримується не лише на персональних комп'ютерах. Цікаво те, що API написано використовуючи концепти технології .NET. [7]

Окрім WinRT все ще існує потужне Win32 API, яке також підтримується на нових версіях Windows 10/11. Для того, аби розробник отримав останні можливості Windows API, не важливо чи він розробляє додаток за моделлю Desktop чи UWP, було створено Project Reunion [7]. Разом з виходом Windows 11 була показана платформа Project Reunion, але під новою назвою – Windows App SDK, про яку і йдеться мова у цій роботі.

Windows App SDK є уніфікованою платформою, яка дозволяє розробляти під Windows 10/11 та для будь-якого пристрою не важливо яку модель розробки обрано – UWP чи Desktop. WinUI 3.0 та MSIX є підпроектами Windows App SDK. [2]

Для розробки застосунку було обрано саме Desktop (Win32) модель, адже для запуску команд додаток повинен мати повний доступ до системи. Крім того, немає змісту підтримувати функціонал застосунку для інших девайсів окрім комп'ютерів, як цього потребує UWP модель. Також використовується відповідна технологія упакування додатку – MSIX-пакет, про який було згадано раніше у пункті 2.1.3.

2.2 C# та .NET

Додаток написано на мові програмування C# з використанням платформи .NET.

Платформа .NET є багатоплатформенним, безкоштовним для використання рішенням від Microsoft з відкритим вихідним кодом для розробки додатків.

У проекті використані остання стабільна версія C# та .NET 7.0.

3 ОСОБЛИВОСТІ РОЗРОБКИ

3.1 MSIX

У пункті 2.1.3 було описано новий спосіб упаковки додатків для Desktop моделі розробки, яка використовується у проекті. Головна перевага використання MSIX технології це те, що програма «живе» у контейнері і там зберігає усі свої дані, що дає можливість операційній системі легко видалити додаток без залишкових файлів. Проте це створює певні обмеження та особливості для розробника.

Основна відмінність полягає у збереженні даних додатку. Система надає виділене сховище для кожного користувача [6]. Розробнику не потрібно знати, де знаходиться це сховище і як у ньому зберігаються дані тому, що операційна система бере відповідальність за такі процеси і забезпечує ізоляцію даних від інших додатків та користувачів. Операційна система також відповідальна за збереження даних під час оновлення програмного забезпечення та за витирання усіх файлів коли користувач видаляє додаток.

Для того, аби зберігати дані використовуються об’єкти *ApplicationDataContainer* та *StorageFolder* [6]. Перший представляє з себе словник, ключем якого є стрічка, а значенням будь-який об’єкт. Це сховище підходить для збереження налаштувань користувача таких, як наприклад вибрана тема (світла/темна). *ApplicationDataContainer* має ліміт на розмір одного запису у 8 кБ, тому часто не підходить для збереження усіх даних програми. Об’єкт *StorageFolder* надає доступ до папки, яка може використовуватись для збереження даних. У папку можна записувати файли різних типів. Обмеження на розмір файлу немає.

Крім того, для збереження даних у Windows App SDK є і інші типи сховищ: для тимчасових файлів та хмарне сховище для синхронізації даних між пристроями, але такі типи сховищ не використовуються у розробленому додатку.

Як уже було згадано раніше MSIX також вміє оновлювати додаток і таким чином спрощує розробнику створення нових версій. Також цей тип пакування використовується у разі завантаження додатку у Microsoft Store.

3.2 MVVM

Раніше підхід до розробки користувацького інтерфейсу полягав у додаванні компонентів і програмної частини для них (code-behind). Проте такий підхід мав ряд недоліків через сильну зв'язність між компонентами та бізнес-логікою. Це ускладнювало тестування функціоналу та додавання нового.

Вирішенням згаданої проблеми є патерн MVVM, який дозволяє відділити бізнес-логіку від користувацького інтерфейсу. Реалізація патерна вимагає створення трьох компонентів як наведено на рисунку 3.1: Model, View і ViewModel.

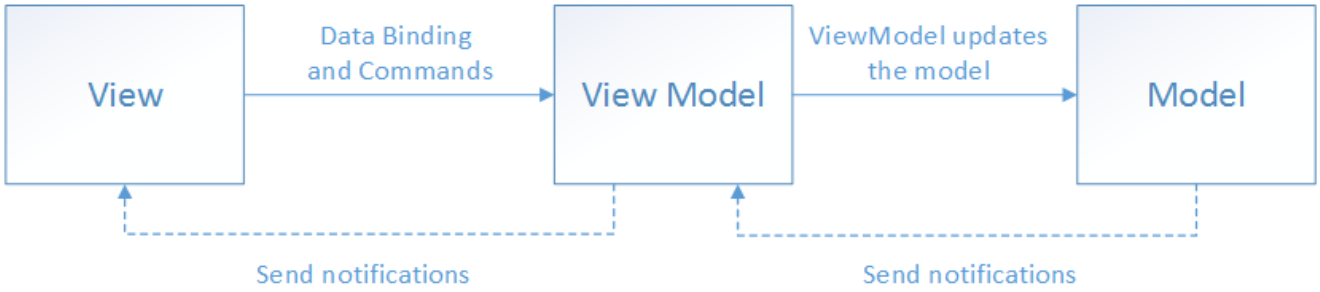

Рисунок 3.1 – Компоненти MVVM патерну та взаємозв'язок між ними.

Такий підхід до розробки є рекомендованим при розробці додатків з використанням Windows App SDK.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ

4.1 Загальна архітектура

Проект складається з двох частин:

1. **Commands** – відповідає за інтерфейс користувача. У ньому визначені компоненти, сторінки та стилі. Проект реалізує MVVM патерн описаний у пункті 3.2 і включає View та відповідні ViewModel компоненти. Він використовує *Commands.Core* компонент.
2. **Commands.Core** – включає в себе бізнес-логіку застосунку. Він відповідає за виконання команд та їх збереження. Цей компонент є незалежним від інтерфейсу користувача і може використовуватись як окремий API модуль.

У компоненті *Commands.Core* визначені моделі даних додатка (див. рисунок 4.1).

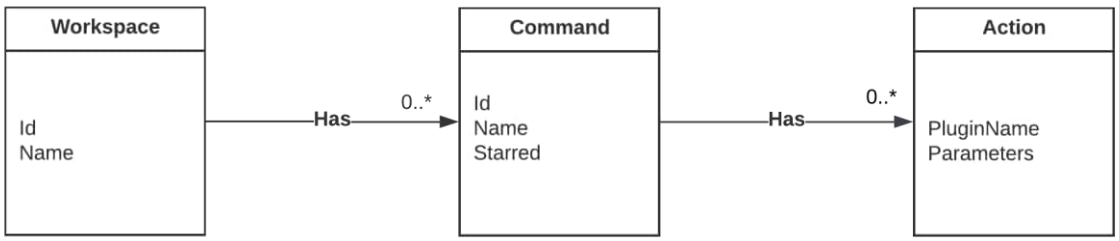

Рисунок 4.1 – Моделі даних, що використані у додатку.

1. **Workspace.** Додаток дозволяє організувати команди у окремі області. Області є аналогами папок файлового провідника. Область має ім’я, унікальний ідентифікатор та масив його команд.

2. **Command.** Команди є головним компонентом програми і зберігають набір дій. Команда має унікальний ідентифікатор, ім’я та прапорець, який позначає, чи вона додана до списку обраних. Також команда включає в себе дії, з яких вона складається.

3. **Action.** Як раніше зазначалось команда складається з дій. Дія репрезентує одиницю роботи. На цей момент у додатку реалізовані наступні дії: «виконати Command Prompt команду» та «виконати PowerShell команду». Проте під час реалізації ми не прив’язувались лише до команд командної стрічки, а зробили модель даних і архітектурну реалізацію універсальною, щоб у майбутньому можна було додати дії іншого типу, наприклад: «Показати вікно введення», «Запустити програму», «Створити файл». Дія містить назву плагіна, який відповідає за її виконання та визначення команди (детальніше описано у пункті 4.2). Також вона включає в себе словник необхідних для виконання команди параметрів.

4.2 Сервіси та «плагіни»

Плагіни це визначення дій, які користувач може використати у командах. Кожен плагін є незалежним від інших і містить необхідні методи для виконання дії за яку він відповідає. Він повинен реалізувати інтерфейс *IActionPlugin,* який містить необхідний набір методів. Інтерфейс містить назву, тип, метод, який повертає значення, чи плагін доступний для користувача, та метод який відповідає за виконання дії, яка передбачена плагіном.

Як вже згадувалось, не кожен плагін доступний для користувача. Це зумовлено тим, що плагіни можуть потребувати додаткових встановлених компонентів. Наприклад, для того, аби дія з PowerShell плагіну була доступна у користувача, він повинен мати встановленим PowerShell на пристрої, де виконується команда. В уже реалізованих у програмі плагінах використовується реєстр значень для того, аби отримати інформацію про те, чи необхідний компонент встановлений і, відповідно, чи дія доступна. Наприклад, PowerShell плагін використовує наступний шлях у реєстрі для перевірки чи відповідна командна стрічка встановлена на комп’ютері: «HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\PowerShell\1»
Такий підхід є найбільш надійним та широко використовується і у інших додатках,

адже ключі реєстру зі значеннями змінюються лише у дуже рідкісних випадках. Проте, такий спосіб перевірки наявності необхідних компонент доступний лише для операційної системи Windows. На даному етапі додаток працює тільки на цій операційній системі, тому це не створює додатковий обмежень.

Для того, аби керувати доступними плагінами створений окремий сервіс під назвою *ActionsService*. Він є менеджером плагінів і відповідає за їх реєстрацію, зберігання і вміє віддавати доступні користувачеві дії.

Також, реалізований окремий сервіс *ActionPluginsInitializer*, який ініціалізує *ActionService*. Цей сервіс є реєстром усіх створених плагінів. Він виконується один раз на етапі запуску програми та реєструє плагіни у *ActionsService*.

Крім того, у додатку є окремий сервіс для управління робочими областями (*WorkspaceDataService*). Він відповідає за управління робочими областями та їхніми командами. Вміє створювати команди, видаляти їх, переміщати між різними областями.

Також, у проекті реалізовані додаткові сервіси, які потрібні для забезпечення зберігання даних (*FileService* та *LocalStorageService*). Детальніше про зберігання даних у додатку викладено у наступному пункті.

Загальна структура компонентів та архітектура додатку зображена на рисунку 4.2.

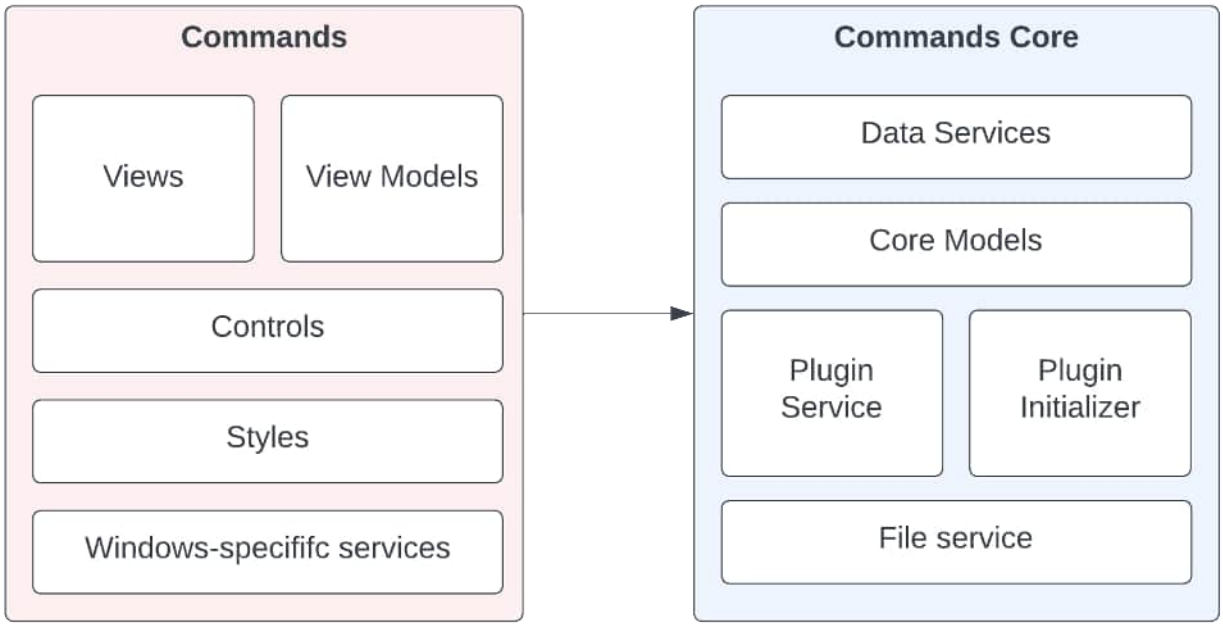

Рисунок 4.2 – Компоненти застосунку.

4.3 Збереження даних

Застосунок повинен зберігати створені команди і налаштування користувача. У пункті 3.1 уже було описано доступні можливості для збереження даних при використанні технології MSIX-упакування. Так як спосіб збереження даних є специфічним для Windows платформи він імплементований у *Commands* компоненті, але інтерфейс функціоналу визначений у *Commands.Core* компоненті, адже у ньому визначені сервіси, які виконують операції збереження і читання даних.

Для збереження налаштувань користувача таких як вибрана тема інтерфейсу використовується *ApplicationDataContainer*.

Команди зберігаються іншим способом, так як їх параметри можуть займати більше місця, ніж дозволяє *ApplicationDataContainer*. Для зберігання таких даних використовується JSON формат. Програмна реалізація зроблена так, що головним об’єктом даних є саме *Workspace*. Тому було прийнято рішення зберігати кілька JSON файлів, кожен з яких відповідає відповідній робочій області (*Workspace*). Для

того, аби зберегти дані використовується механізм серіалізації даних. Усі файли даних робочих областей зберігаються в окремій папці. Доступ до папки надається уже згаданим раніше у пункті 3.1 об’єктом *StorageFolder*. При запуску програми сканується уся, виділена для даних робочих областей папка, і на основі кожного знайденого JSON файла створюється нова робоча область з доданими до неї командами. Загальна структура файлу конфігурації робочої області наведена на рисунку 4.3.

C: > Users > omane > AppData > Local > Packages > f9a09f4f-2321-4c3b-aafc-48c8e02678fe_ezndb9mcqcjar > LocalState > {} Default.json > ...

```json
{
  "Id": "41d3a042-a9c4-4b43-bf6a-ff29cf1c941f",
  "Name": "Default",
  "Commands": [
    {
      "Id": "dfba5b6f-6bd0-4101-bc6b-3ca9f829f1dd",
      "Name": "Build solution",
      "Starred": false,
      "Actions": [
        {
          "PluginName": "CommandPrompt",
          "Parameters": { "Script": "dotnet build", "KeepShowWindow": true }
        },
        {
          "PluginName": "PowerShell",
          "Parameters": { "Script": "dotnet restore", "KeepShowWindow": true }
        }
      ]
    },
    {
      "Id": "c3307f77-cd82-484f-adec-067149a2bcd8",
      "Name": "Run frontend tests",
      "Starred": true,
      "Actions": [
        {
          "PluginName": "CommandPrompt",
          "Parameters": {
            "Script": "cd /client/ & pnpm run test",
            "KeepShowWindow": true
          }
        }
      ]
    }
  ]
}
```

Рисунок 4.3 – Файл конфігурації робочої області.

Такий спосіб збереження даних також надає легкий спосіб поширення команд для інших користувачів. Для цього достатньо надіслати JSON файл потрібної робочої області на інший пристрій і там додати його у правильну папку.

Для серіалізації та десеріалізації даних, а також читання і записування файлів використовується окремий сервіс, визначений у *Commands.Core* компоненті – *FileService*.

4.4 Огляд застосунку

Після запуску програми відкривається головне вікно яке показано на рисунку 4.4. З лівого боку знаходиться панель навігації. Її можна розгорнути натиснувши на кнопку для розгортання меню або збільшивши ширину вікна. Панель навігації зображена на рисунку 4.5. Для відображення меню використовується компонента *NavigationView*. З бічного меню є можливість відкрити домашню сторінку, сторінку робочої області та сторінку налаштувань. Також доступна функціональність пошуку команд.

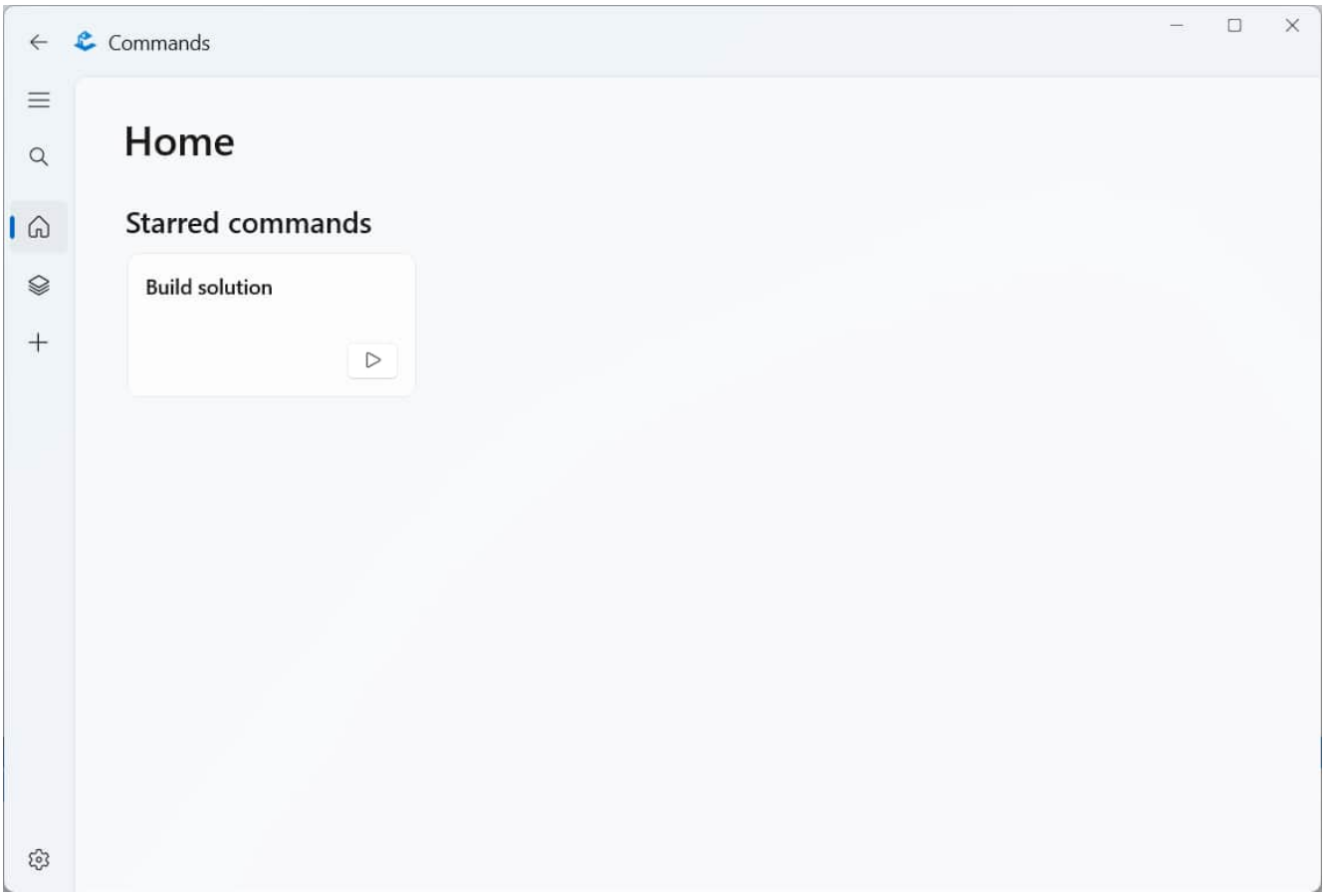

Рисунок 4.4 – Головне вікно програми.

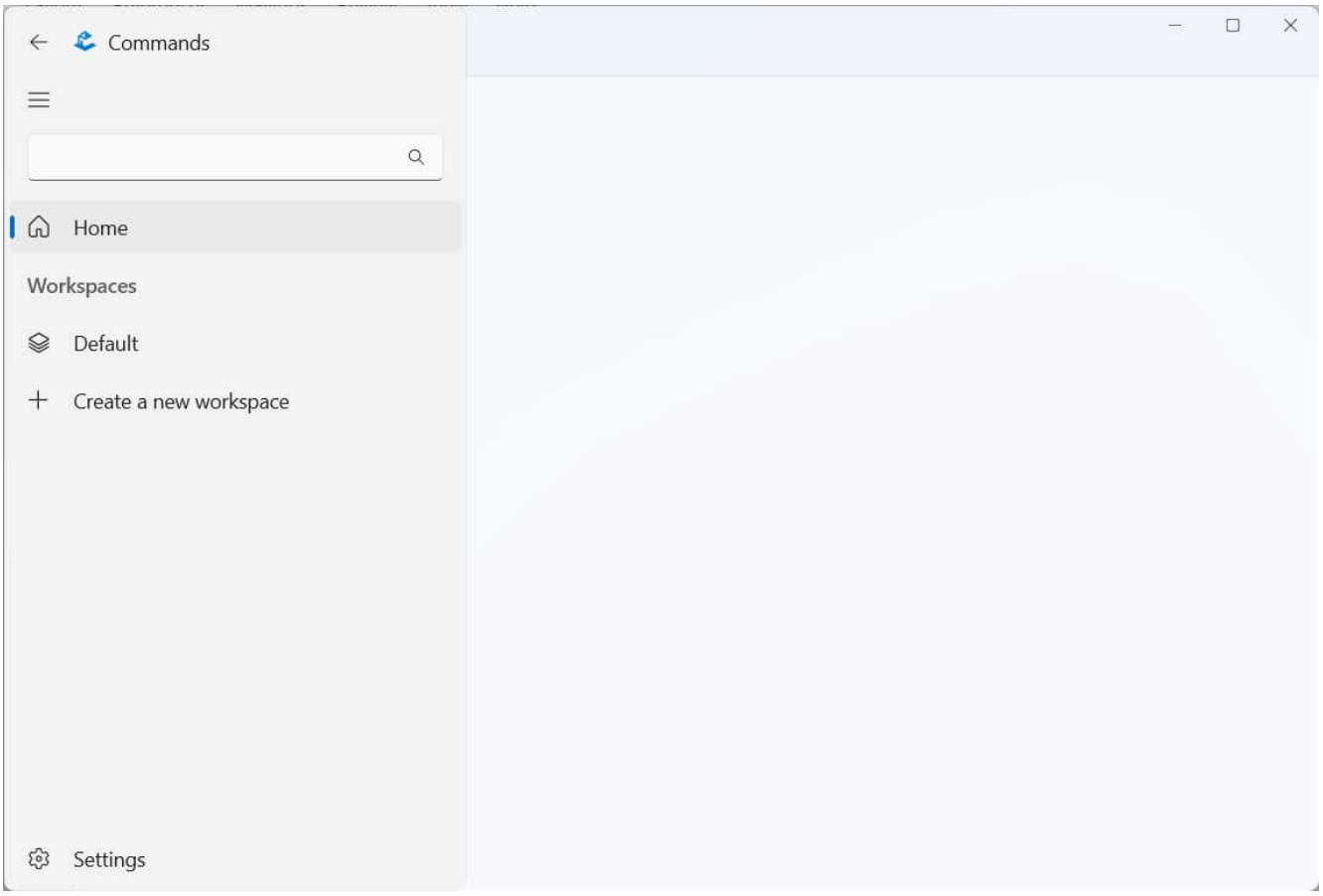

Рисунок 4.5 – Меню програми.

Сторінка «Home» (домашня сторінка) показана на рисунку 4.4. Вона з’являється першою, після запуску програми. На даний момент там відображаються команди, які додані до списку вибраних.

Під сторінкою «Home» знаходиться список робочих областей. За замовчуванням у користувача буде створена робоча область з назвою «Default». Також є кнопка «Create a new workspace» для створення нової робочої області. Після натискання з’являється діалог, як показано на рисунку 4.6.

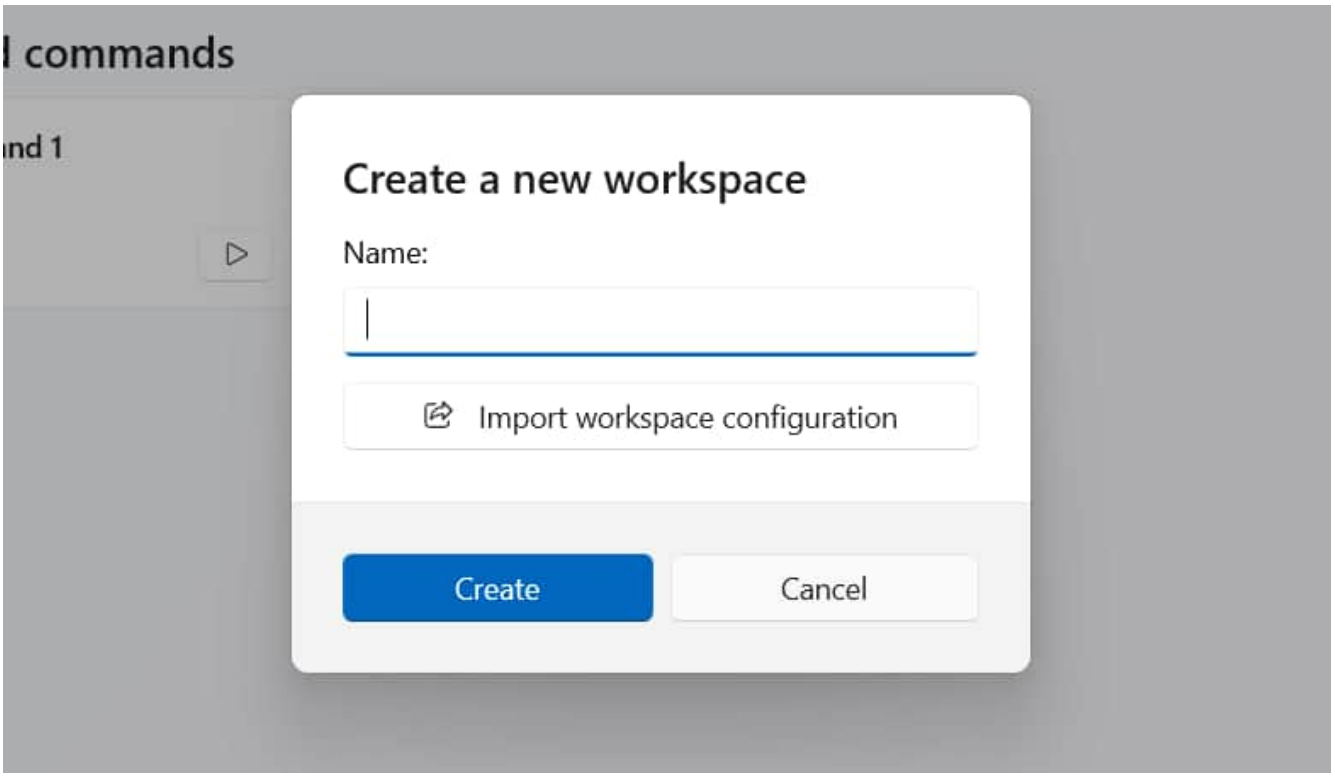

Рисунок 4.6 – Діалог створення нової робочої області.

Після переходу до будь-якої з створених робочих областей з’являються кнопки для додавання нової команди і для поширення файлу конфігурації робочої області, це можна побачити на рисунку 4.7. Після натискання кнопки для поширення відкривається JSON файл цієї робочої області, структура якого раніше вже була згадана у пункті 4.3. При створенні нової робочої області є можливість імпортувати цей JSON файл. Для цього використовується кнопка «Import workspace configuration» (див. рисунок 4.6), яка відкриває вікно файлового провідника для вибору файлу. Таким чином реалізовано можливість поширення команд для інших користувачів. На даному етапі усі команди та дії, що містяться у файлі конфігурації імпортуються без виконання валідації того, чи необхідні для виконання плагіни доступні у користувача, що виконує імпорт. Цей процес буде покращено у наступних версіях додатку.

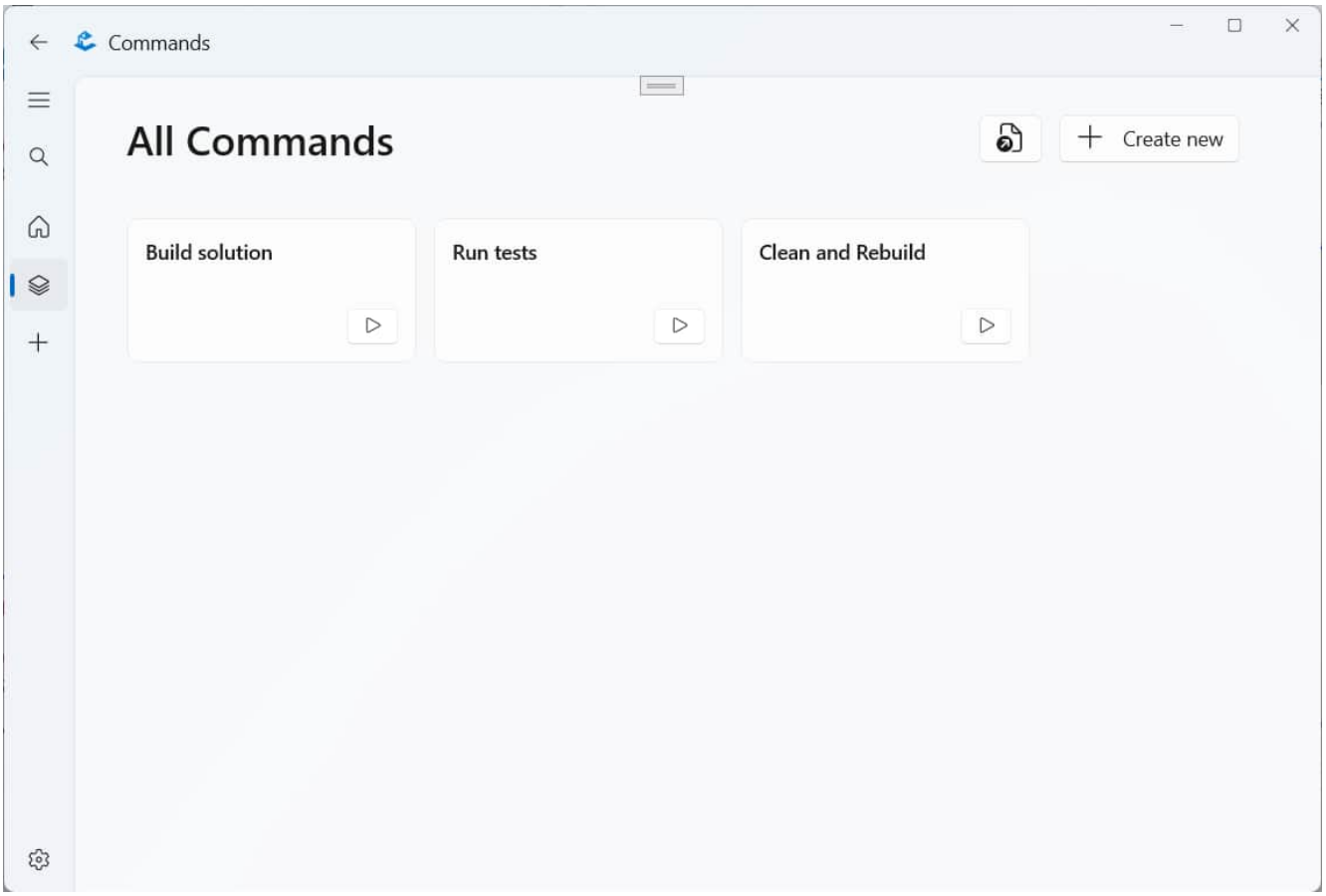

Рисунок 4.7 – Команди робочої області.

Вигляд сторінки редагування команди показаний на рисунку 4.8. На сторінці знаходиться поле для редагування назви команди, кнопка для запуску команди, додаткове меню (кнопка з трьома крапками) та область з доданими до команди діями. За допомогою кнопки «Add new action» додаються нові дії. Після натискання на неї з’являється список доступних дій. Додані дії мають різні параметри які можна задати: в даному випадку це стрічка команди та перемикач, який відповідає за те, чи буде вікно командного рядка закриватись після виконання команди (дії).

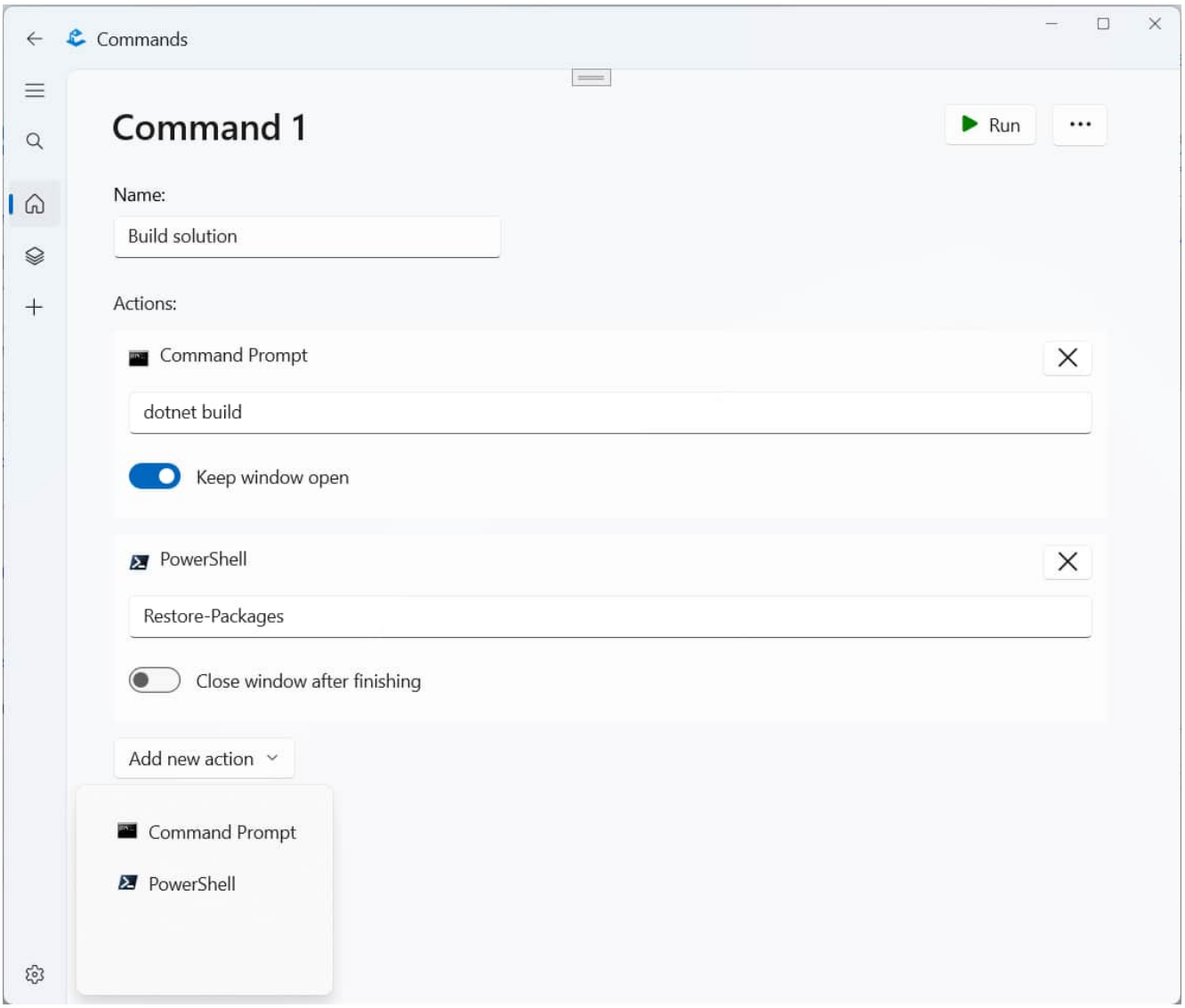

Рисунок 4.8 – Сторінка редагування команди.

У додатковому меню можна додати команду до обраних, після чого вона з’явиться на домашній сторінці, перемістити у іншу робочу область та видалити команду. Додаткове меню показане на рисунку 4.9.

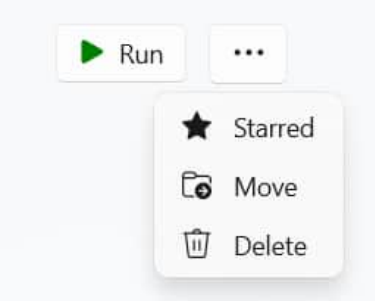

Рисунок 4.9 – Додаткове меню команди.

Також, з меню додатку можна перейти до налаштувань. Сторінка налаштувань зображена на рисунку 4.10. Користувач може змінити тему оформлення інтерфейсу на світлу або темну, вигляд яких показано на рисунках 4.7 та 4.11 відповідно. Також можна використовувати стандартну тему, в такому випадку використовується та тема, яка задана у операційній системі. Функціонал зміни теми оформлення інтерфейсу є стандартною можливістю додатків, що використовують Windows App SDK. Усі компоненти WinUI адаптовані для використання у світлих та темних оформленнях графічного інтерфейсу користувача.

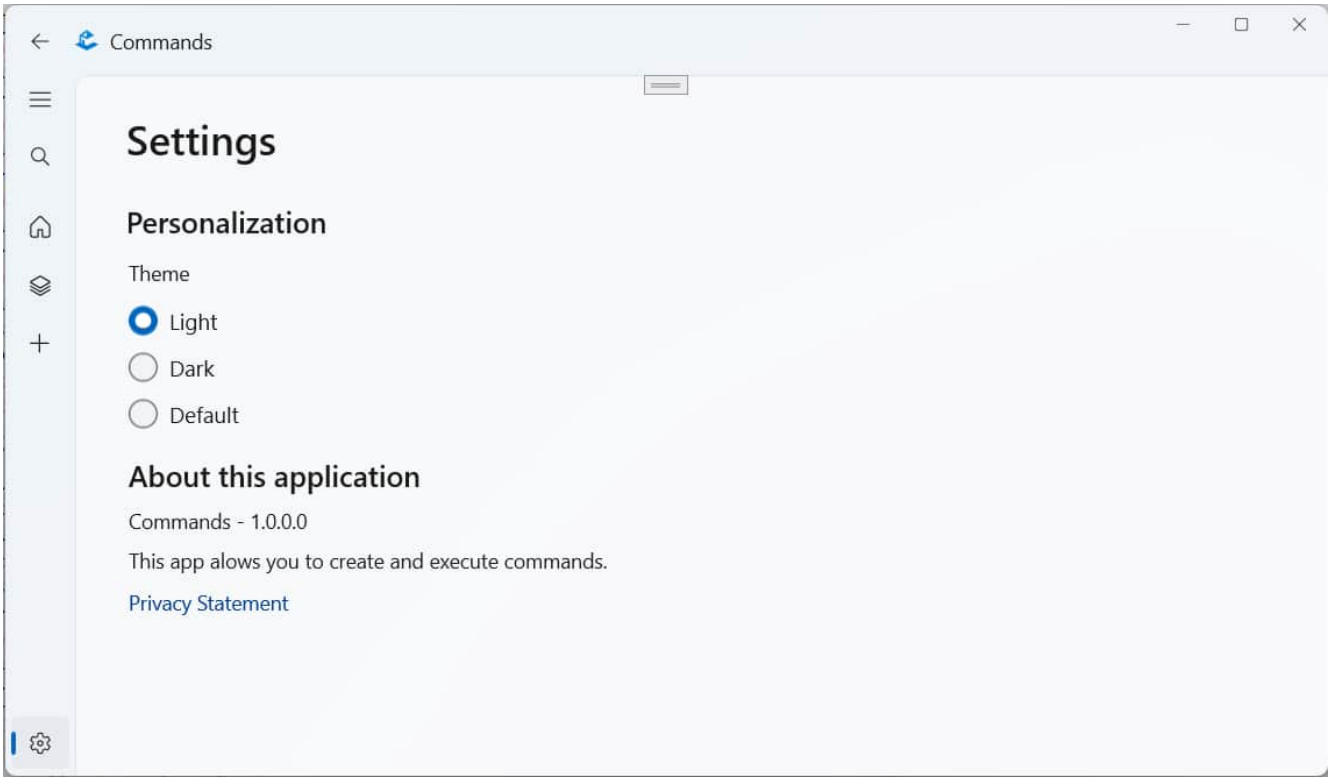

Рисунок 4.10 – Сторінка налаштувань додатку.

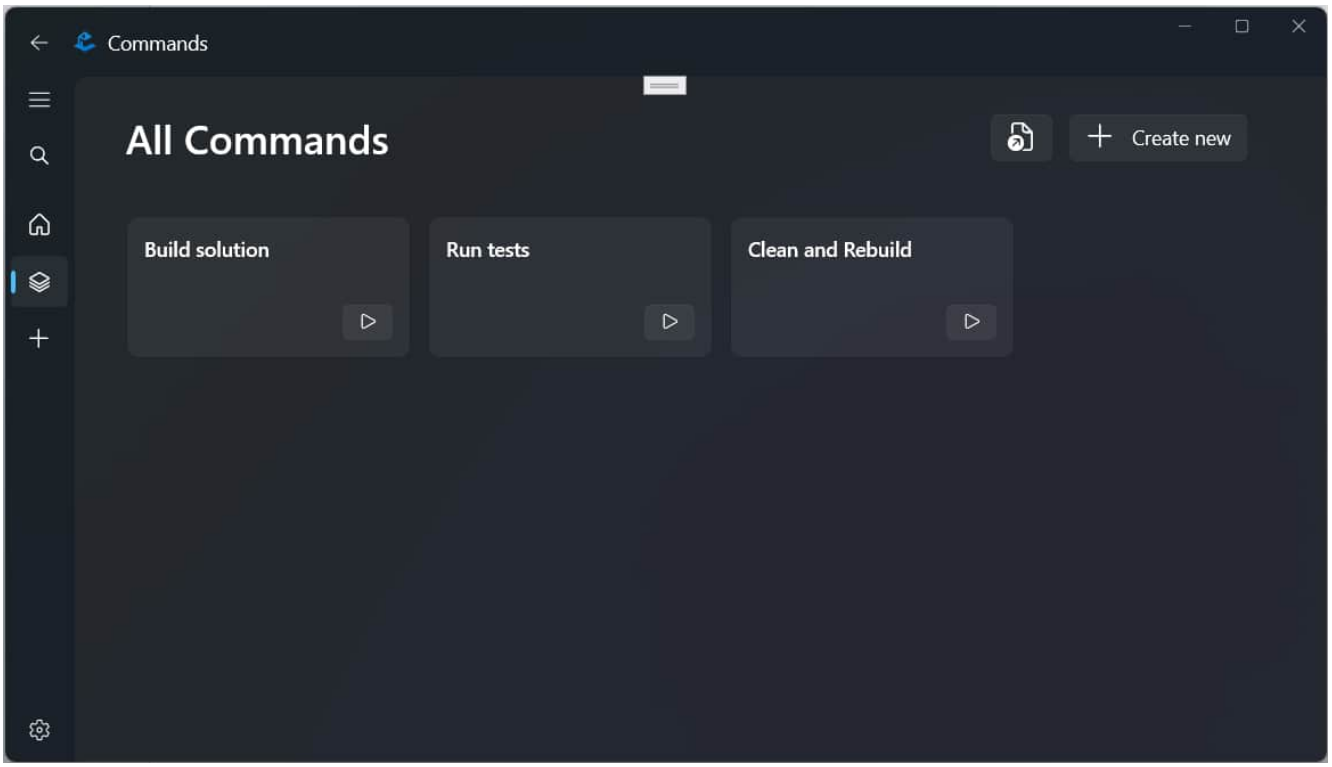

Рисунок 4.11 – Темна тема оформлення інтерфейсу.

4.5 Запуск команд

Після запуску команди послідовно виконуються усі додані до неї дії. Усі доступні зараз дії виконуються з допомогою *System.Diagnostics.ProcessStartInfo*. Він дозволяє запускати процеси і передавати аргументи для їх запуску. У випадку Command Prompt та PowerShell команд аргументами запуску є стрічка команди, яку потрібно виконати.

Виконання дій відбувається у окремих вікнах, але у майбутньому планується відображати результати роботи у самому додатку. В процесі реалізації на даному етапі з цим виникли певні труднощі, адже один процес може створювати інші, нові процеси. Це значно ускладнює збір результатів роботи дії.

ВИСНОВКИ

У цій роботі було створено додаток для збереження команд командного рядка. Додаток дозволяє додавати дії до команди, такі як: «Виконати Command Prompt команду» та «Виконати PowerShell команду». Одна команда може містити кілька дій різного типу. Архітектура додатку побудована на основі плагінів для виконання дій. Це дозволяє легко та швидко розширювати функціональність у майбутньому.

Також, була передбачена можливість організовувати команди у окремі області та поширювати їх для інших користувачів. Це важлива частина функціональності додатку, адже вона дозволяє додавати файли конфігурацій у репозиторії чи документації, а користувач може легко завантажити їх собі, що значно економить час, так як команди не потрібно створювати самостійно. Крім цього, реалізована можливість зберігання команд до списку обраних для швидкого доступу до них.

Одним з важливих аспектів цієї роботи є також дослідження можливостей Windows App SDK. Через те, що технологія є досить новою, наразі існує дуже невелика кількість матеріалів для вивчення особливостей роботи з нею. Тому інтерфейс користувача та функціонал створювались за загальними правилами та рекомендаціями до написання десктопних Windows-додатків і з використанням офіційної документації компанії Microsoft.

Windows App SDK надає для використання потужне і сучасне Windows API. Однією з компонент SDK, яка використана у створеному додатку є MSIX упакування. Це дозволяє легко встановлювати додаток, оновлювати та видаляти його. Також, використання цієї технології дає можливість розповсюджувати додаток через Microsoft Store. Проте, використання Windows App SDK створює і певні обмеження. Наприклад, додаток може працювати лише на операційних системах Windows 10/11.

Для реалізації додатку також були використані C#, .NET та WinUI. Використання сучасної бібліотеки компонентів та стилів WinUI, яка є частиною Windows App SDK надало можливість створити передовий додаток, який побудований на елементах інтерфейсу операційної системи. Також це дозволило реалізувати перемикання між темами оформлення інтерфейсу, так як усі компоненти бібліотеки підтримують таку функціональність, а це підвищує комфорт від використання застосунку.

Варто також зазначити, що подібних публічних додатків для операційної системи Windows немає. Це вимагало створення додатку з повного нуля без можливості попереднього аналізу уже наявних додатків.

Отже, у результаті виконання дипломної роботи вдалось створити застосунок, що виконує раніше описані вимоги і дозволяє пришвидшити та спростити процес розробки для розробників програмного забезпечення.

Список використаних джерел

1. .NET Documentation [Електронний ресурс] // Microsoft Documentation. – 2023. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/fundamentals/.
2. Windows App SDK Documentation [Електронний ресурс] // Microsoft Documentation. – 2023. – Режим доступу: https://learn.microsoft.com/en-us/windows/apps/windows-app-sdk/.
3. Windows UI Library in the Windows App SDK (WinUI 3) [Електронний ресурс] // Microsoft Documentation. – 2023. – Режим доступу: https://learn.microsoft.com/en-us/windows/apps/winui/winui3/.
4. Universal Windows Platform documentation [Електронний ресурс] // Microsoft Documentation. – 2023. – Режим доступу: https://learn.microsoft.com/en-us/windows/uwp/.
5. Design and code Windows apps [Електронний ресурс] // Microsoft Documentation. – 2023. – Режим доступу: https://learn.microsoft.com/en-us/windows/apps/design/.
6. Windows Runtime API [Електронний ресурс] // Microsoft Documentation. – 2023. – Режим доступу: https://learn.microsoft.com/en-us/uwp/api/.
7. What is Actually the Universal Windows Platform and what is WinUI, MSIX, and Project Reunion / Windows App SDK? [Електронний ресурс] // Thomas Claudius Huber. – 2021. – Режим доступу: https://www.thomasclaudiushuber.com/2021/02/05/what-is-actually-the-universal-windows-platform-and-what-is-winui-msix-and-project-reunion/.
8. Troelsen A. W. Pro C# 7 With .NET and .NET Core / A. W. Troelsen, P. Japikse. – Apress, 2017. – 1372 p.
9. JSON [Електронний ресурс] // MDN web docs – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON.

10. WinUI Gallery [Електронний ресурс] // Microsoft – Режим доступу до ресурсу: https://github.com/microsoft/WinUI-Gallery.
11. Intro to Shortcuts on Mac [Електронний ресурс] // Apple – Режим доступу до ресурсу: https://support.apple.com/en-gb/guide/shortcuts-mac/apdf22b0444c/mac.

Додаток А. Застосунок Команди для операційних систем MacOS

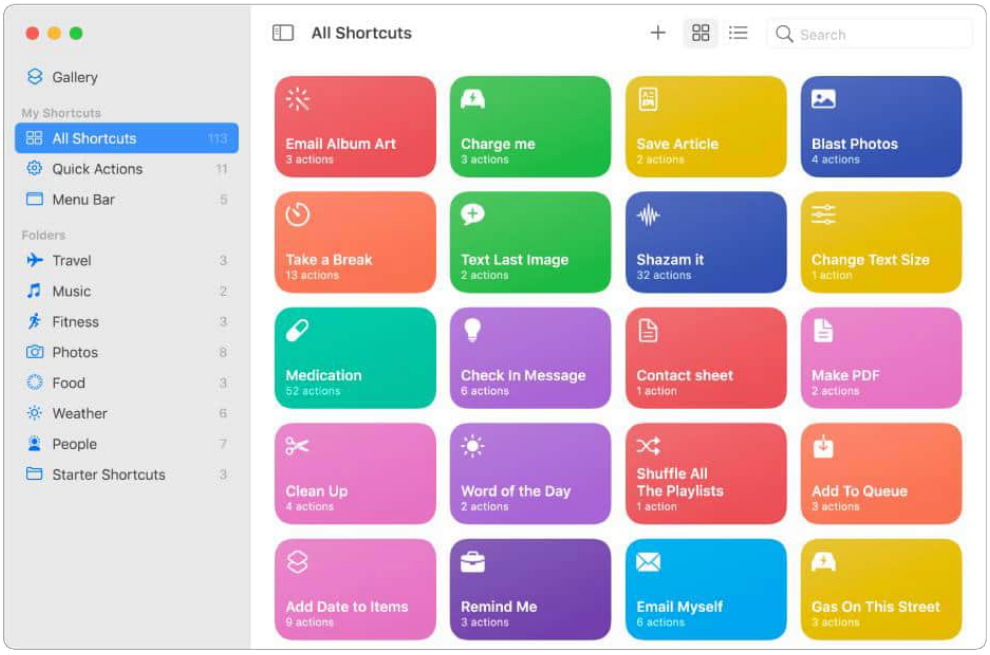

Рисунок А.1 - Список створених команд

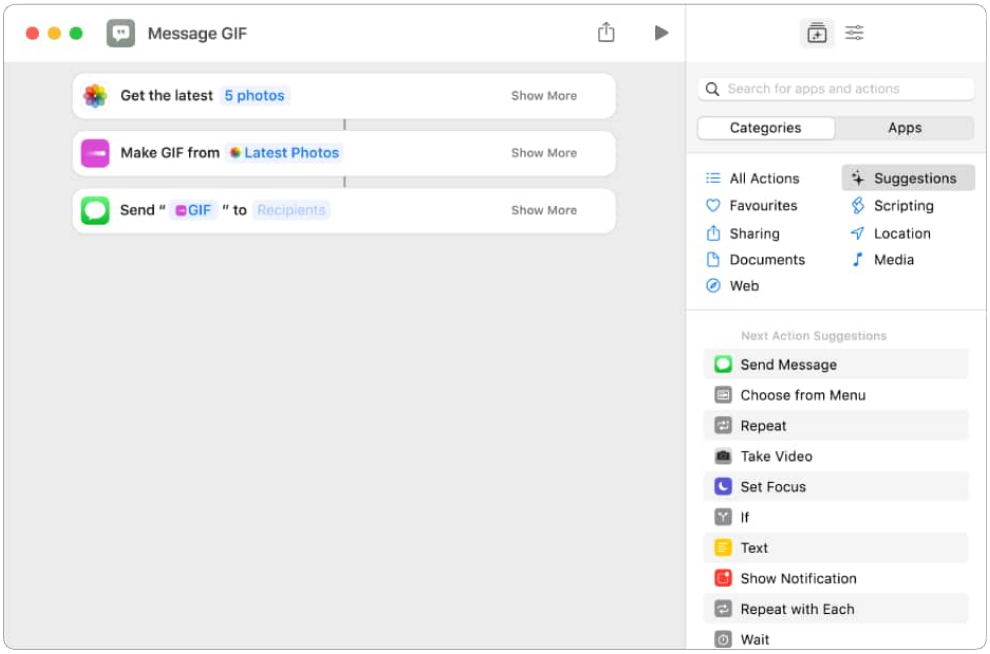

Рисунок А.2 – Вікно редагування команди